Introduction to the Abstract Meaning Representation (AMR)

http://tiny.cc/amrtutorial

http://amr.isi.edu/

Why abstract?

- English provides many ways to express even simple ideas.
    - Too many to simply write down a few rules to characterize, e.g., paraphrase alternations.
- For many NLP applications, we want to abstract away from the details of English grammar.
    - What is deeper than syntax? Semantics!

But hasn’t this been done before?

- Long tradition in linguistics and CL of formalizing semantics.
- The key insights behind AMR:

(1) statistical NLP needs a semantic representation that is **practical for large-scale human annotation** (sembanking)

- ▪ What is practical? **limited canonicalization**

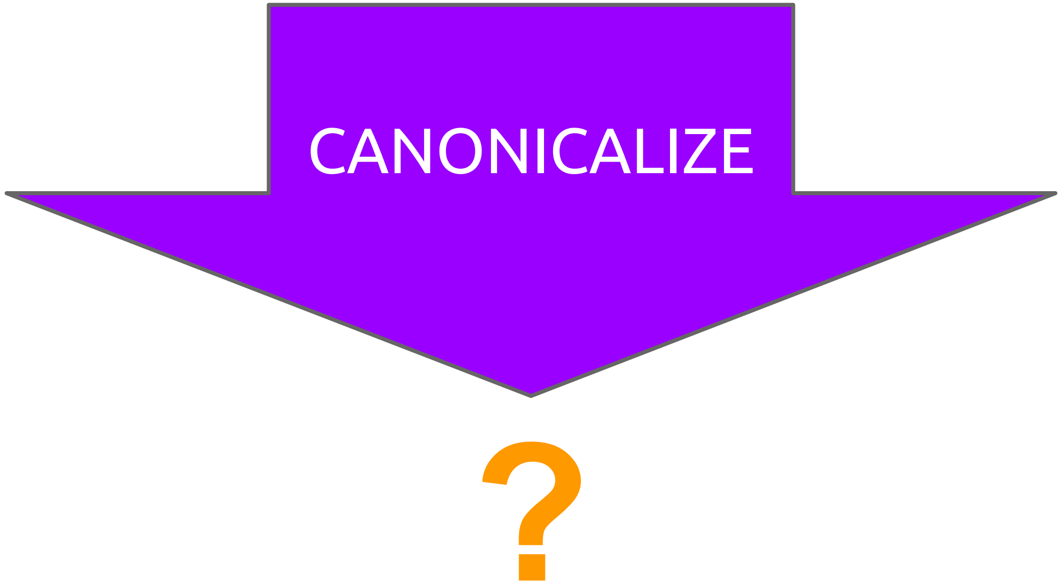
(2) many crucial aspects of meaning can be captured with broad coverage **in a single data structure**

The man described the mission as a disaster.

The man's description of the mission: disaster.

As the man described it, the mission was a disaster.

The man described the mission as disastrous.

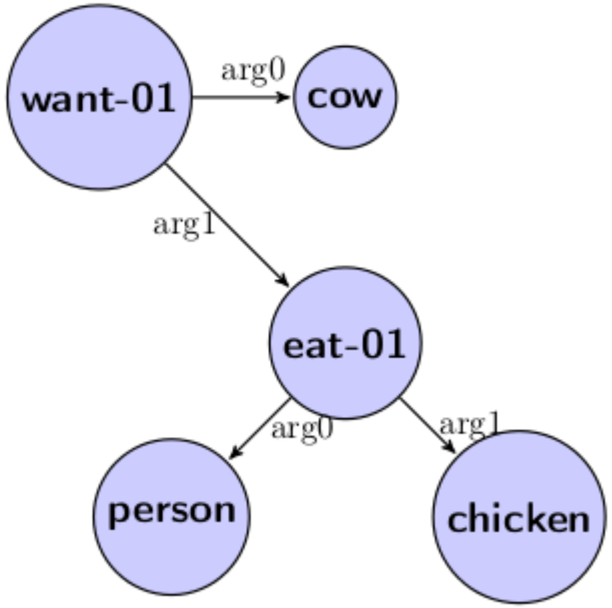


Roadmap for Part I of the tutorial

- Fundamentals of the representation
  - how AMR graphs are structured to represent concepts and relations
- Hands-on annotation practice
  - the annotation tool, simple examples
- Survey of linguistic/semantic phenomena
- Comparison to other representations
- Annotation practice
  - more realistic examples

AMR representation itself

"Cows want people to eat chicken."



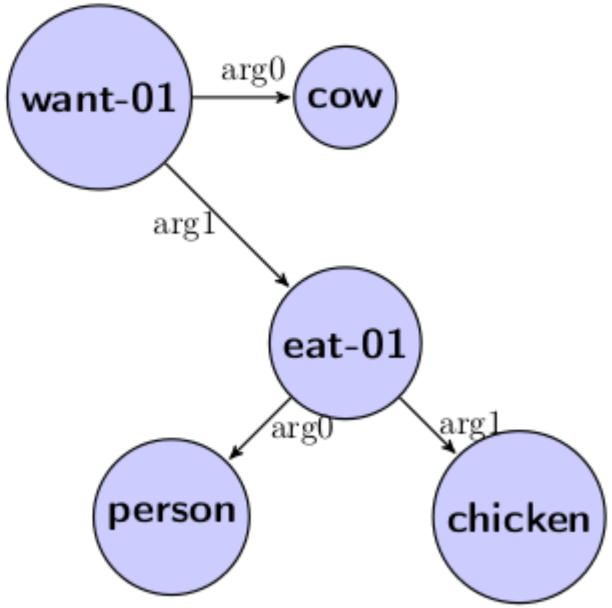
=

```
(w / want-01
      :ARG0 (c / cow)
      :ARG1 (e / eat-01
            :ARG0 (p / person)
            :ARG1 (c2 / chicken)))
```

You can think of each variable as a unique node in a graph
"c / cow" means "c is an instance of the concept cow"

AMR representation itself

"Cows want people to eat chicken."



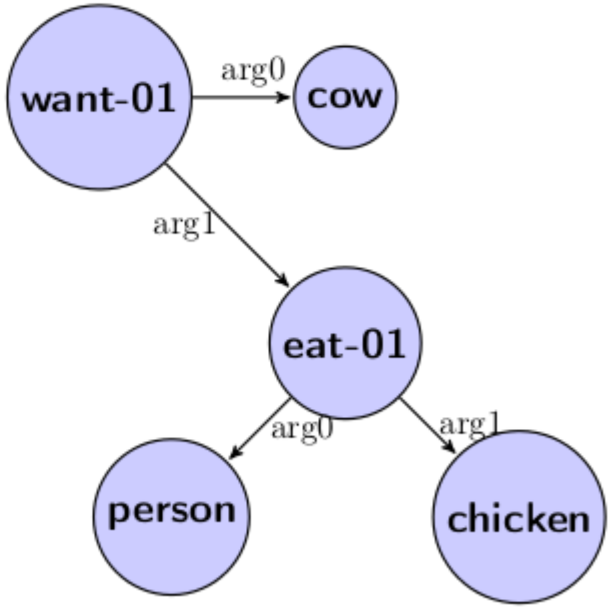
=

```
(w / want-01
      :ARG0 (c / cow)
      :ARG1 (e / eat-01
            :ARG0 (p / person)
            :ARG1 (c2 / chicken)))
```

Some of the nodes are sense disambiguated (the predicates), dropping tense and aspect.
Non-predicative, unnamed concepts are often left as stemmed words, dropping plurality

AMR representation itself

"Cows want people to eat chicken."



=

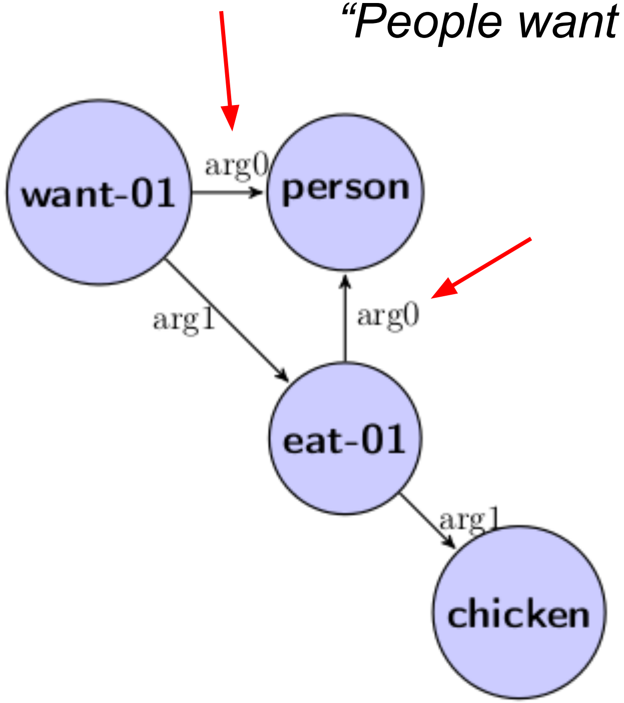
```
(w / want-01
      :ARG0 (c / cow)
      :ARG1 (e / eat-01
            :ARG0 (p / person)
            :ARG1 (c2 / chicken)))
```

Relations are preceded by colons, and can be numbered arguments or more general relations.

The general tendency is that ARG0 is the agent and ARG1 the undergoer.

AMR representation itself

"People want to eat chicken."



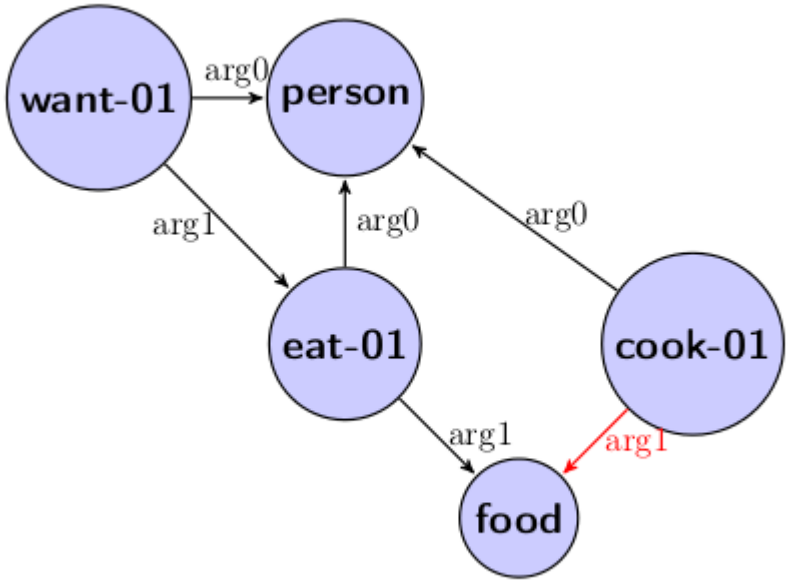
=

```
(w / want-01
      :ARG0 (p / person)
      :ARG1 (e / eat-01
            :ARG0 p
            :ARG1 (c2 / chicken)))
```

Concepts can have more than one semantic role.
In this Penman format, we represent that with "reentrancies" like p

AMR representation itself

"People want to eat food that they cooked themselves"



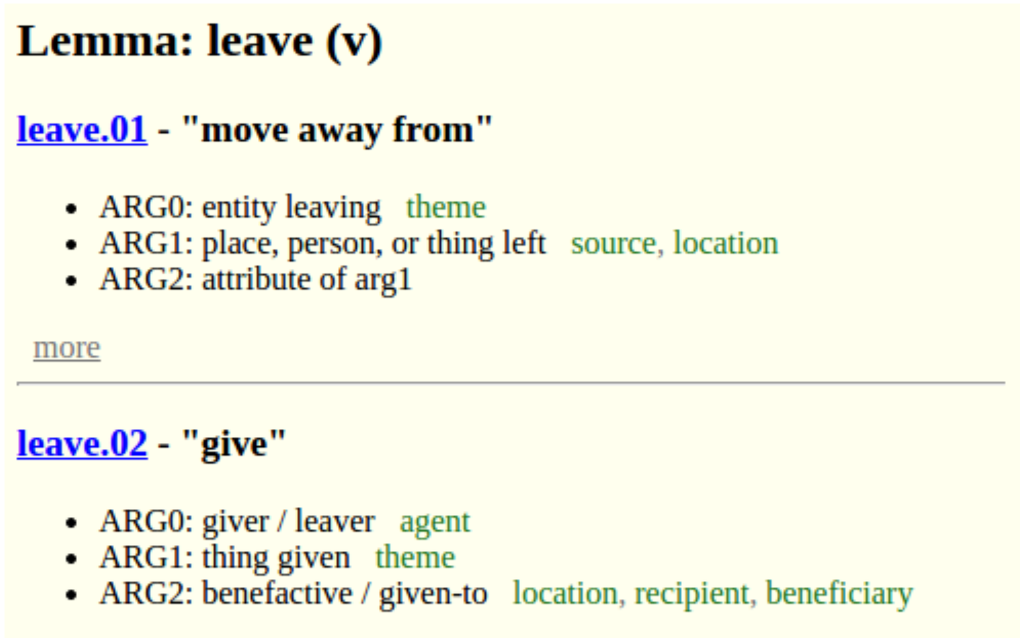
=

```
(w / want-01
    :ARG0 (p / person)
    :ARG1 (e / eat-01
        :ARG0 p
        :ARG1 (f / food
            :ARG1-of (c / cook-01
                :ARG0 p))))
```

For things like relative clauses, the predicate modifies the term using an inverse relation, arg1-of. **This is just a notational trick to represent the graph as a tree.**

Lexicon: what “want-01” and “arg0” mean

- AMR concepts are not the same as strings!
- We use an inventory of conceptual frames: the unified PropBank rolesets.
- The numbered (core) semantic roles are specific to each roleset.



Lemma: leave (v)

leave.01 - "move away from"

- ARG0: entity leaving theme
- ARG1: place, person, or thing left source, location
- ARG2: attribute of arg1

more

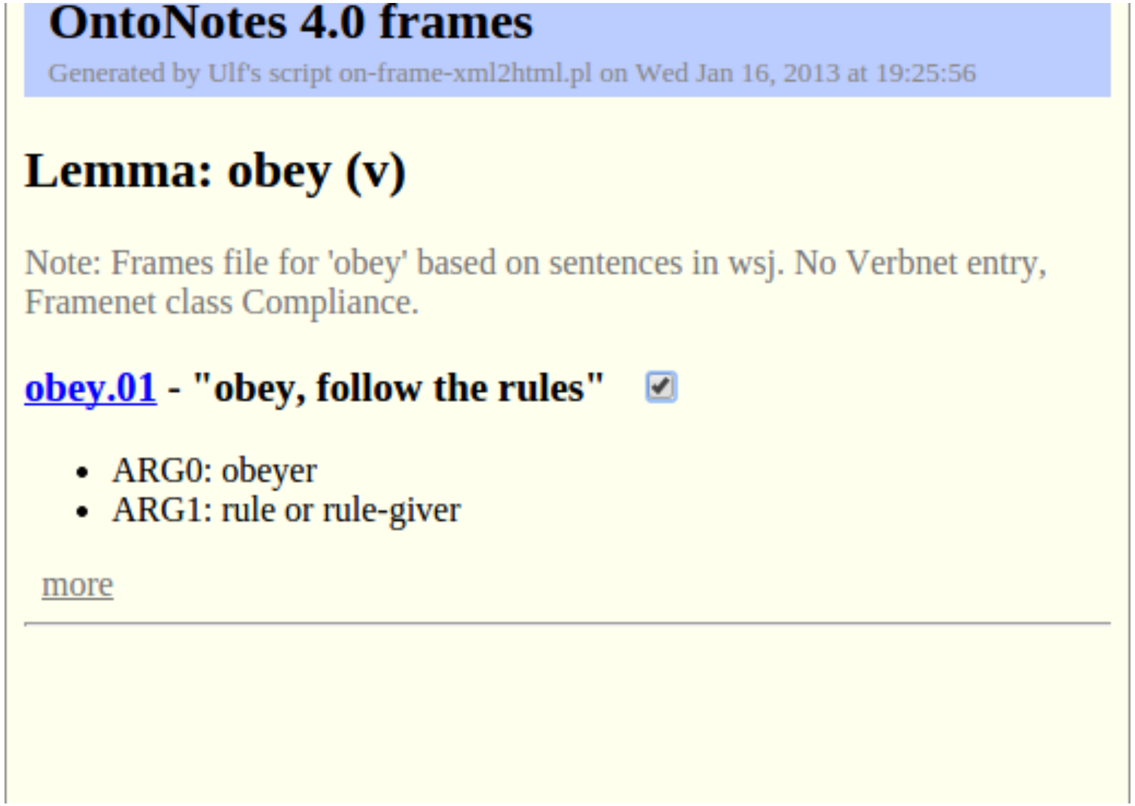
leave.02 - "give"

- ARG0: giver / leaver agent
- ARG1: thing given theme
- ARG2: benefactive / given-to location, recipient, beneficiary

Lexicon: what “want-01” and “arg0” mean

Annotators see
a list that shows
all possible rolesets
and what each numbered
argument means.

There are PropBank
numbered arguments
(what you see in
Ontonotes)



OntoNotes 4.0 frames

Generated by Ulf's script on-frame-xml2html.pl on Wed Jan 16, 2013 at 19:25:56

Lemma: obey (v)

Note: Frames file for 'obey' based on sentences in wsj. No Verbnet entry, Framenet class Compliance.

obey.01 - "obey, follow the rules" ✓

- ARG0: obeyer
- ARG1: rule or rule-giver

more

Lexicon: What it doesn't cover

- This only applies to predicates!
- Non-predicative terms are not sense-disambiguated.
  (Excluding named entities)

```
(o / obey-01
      :ARG0 (w / we)
      :ARG1 (l / law
            :topic (t / thermodynamics))
      :location (h / house
            :mod (t2 / this)))
```

Lexicon: what “want-01” and “arg0” mean

- These rolesets contain many parts of speech:

fear-01

My fear of snakes
I am fearful of snakes
I fear snakes
I’m afraid of snakes

BUT we only link between the same sense when etymologically related.

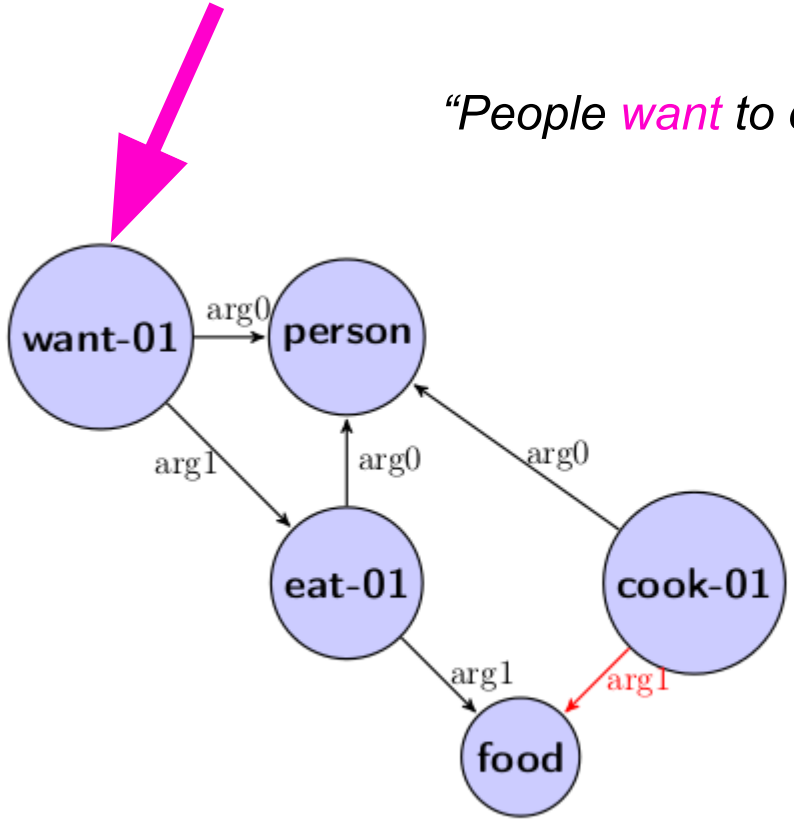
fear-01 ✗

I’m terrified of snakes
Snakes creep me out

Focus

- The **focus** is the concept at the top of the annotation.
  - Must be a root (no incoming edges).
- Which concept should focused?
  - Conceptually, the main assertion of (the declarative version of) the sentence.
  - Linguistically, usually the main predication of the sentence.

Focus



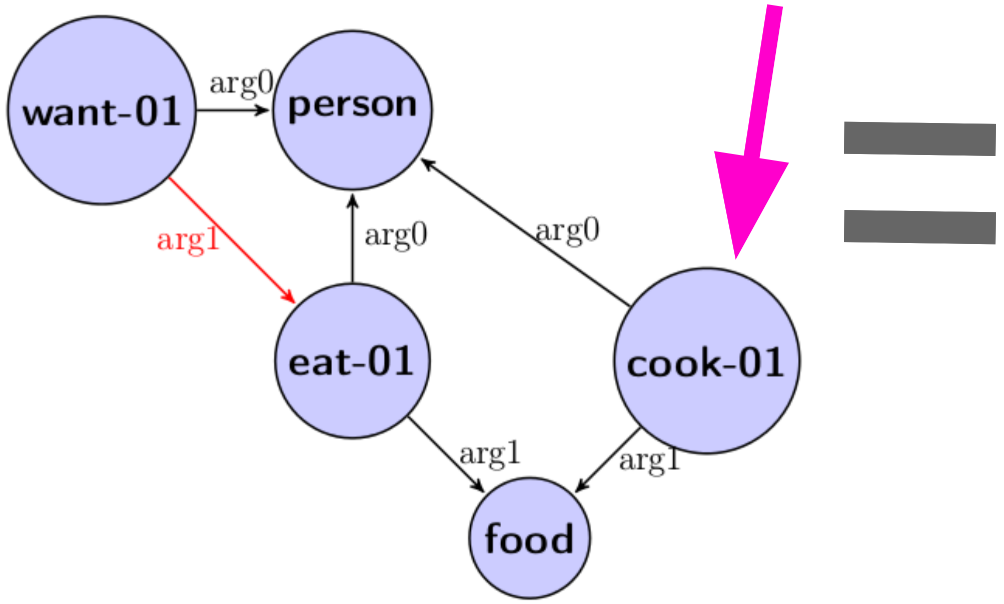
"People want to eat food that they cooked themselves"

=

```
(w / want-01
    :ARG0 (p / person)
    :ARG1 (e / eat-01
        :ARG0 p
        :ARG1 (f / food
            :ARG1-of (c / cook-01
                :ARG0 p))))
```

Focus

"People cook the food that they want to eat themselves"



```
(c / cook-01
    :ARG0 (p / person)
    :ARG1 (f / food
          :ARG1-of (w / want-01
             :ARG0 p
             :ARG1 (e / eat-01
                     :ARG0 p)))
```

Focus

“People want to eat food that they cooked themselves”

```
(w / want-01
    :ARG0 (p / person)
    :ARG1 (e / eat-01
        :ARG0 p
        :ARG1 (f / food
            :ARG1-of (c / cook-01
                :ARG0 p))))
```

“People cook the food that they want to eat themselves”

```
(c / cook-01
    :ARG0 (p / person)
    :ARG1 (f / food
        :ARG1-of  (w / want-01
            :ARG0 p
            :ARG1 (e / eat-01
                :ARG0 p)))
```

Propositionally, these are the same! But different emphasis.

Attribution of properties

Depending on focus, we use special roles `:mod` or `:domain` (these are inverses of each other).

the big house
There is a big house.

```
(h / house
    :mod (b / big))
```

The house is big.

```
(b / big
    :domain (h / house))
```

seeing the big house
seeing the house that is big

```
(s / see-01
    :ARG1 (h / house
                   :mod (b / big)))
```

seeing that the house is big

```
(s / see-01
    :ARG1 (b / big
                   :domain (h / house)))
```

Attribution of properties

Also for attributive/predicative demonstratives and nominals:

this house

```
(h / house
    :mod (t / this))
```

this is a house

```
(h / house
    :domain (t / this))
```

a monster truck

```
(h / truck
    :mod (m / monster))
```

the truck is a monster

```
(m / monster
    :domain (t / truck))
```

Canonicalization

```
(d / describe-01
   :arg0 (m / man)
   :arg1 (m2 / mission)
   :arg2 (d / disaster))
```

The man described the mission as a disaster.

The man's description of the mission: disaster.

As the man described it, the mission was a disaster.

The man described the mission as disastrous

AMR design principles:
Morphosyntactic sugar is considered unhealthy.

Deep is better than shallow. (Paraphrases should have the same AMR.)

Non-core Roles

- Non-core arguments: not predicate-specific (not listed in lexicon)
- The boy wanted to go yesterday

```
(w / want-01
    :arg0 (b / boy)
    :arg1 (g / go-01
        :arg0 b)
        :time (y / yesterday))
```

Non-core Roles

- Relations that aren’t predicate-specific are handled with a large inventory of non-core semantic roles.

:time	:location	:purpose	:frequency
:destination	:subset	:part	:manner
...and many more! The full lists is in the handout			

Non-core Roles

- There is also another kind of numbered argument for things where the number means nothing other than order: **op#**
- Apples and Bananas

```
(a / and
    :op1 (a2 / apple)
    :op2 (b / banana))
```

Non-core Roles

- There is also another kind of numbered argument for things where the number means nothing other than order: **op#**
- Competition between lions, tigers and bears

```
(b / between
    :op1 (l / lion)
    :op2 (t / tiger)
    :op3 (b2 /bear))
```

Constants

String	Numeric

`name :op1 "Yoda"`
`:time "16:30"`
	`:quant 5`

Named	**+/-**

`monetary-quantity :unit dollar`
`:mode imperative`
	`:polarity -`
	`:polite +`

Constants vs. Concepts

- A **concept** is a type. For every concept node there will be ≥1 instance variable/node.
  - An instance can be mentioned multiple times.
  - Multiple instances of the same concept can be mentioned.
- **Constants** are singleton nodes: no variable, just a value. Specific non-core roles allow constant values.

Negation

I am not a crook.

```
(c / crook
     :domain (i / i)
     :polarity -)
```

Negation

Negation goes where it is logical:

I don’t believe we’ve met.

(*meaning:* ‘I believe we haven’t met.’)

```
(b / believe-01
    :ARG0 (i / i)
    :ARG1 (m / meet-02 :polarity -
              :ARG0 (w / we)))
```

Negation by morphology

an unhappy cat

```
(c / cat
     :mod (h / happy :polarity -))
```

illegible writing

```
(t / thing
    :ARG1-of (w / write-01
                :manner (l / legible :polarity -)))
```

Entities

- general concepts are simply stemmed (drop plurality, articles): “the boys” → (b / boy)

“Viacom”

```
(c / company
   :name (n / name :op1 “Viacom”))
```

“Barack Obama”

```
(p / person
   :name (n / name :op1 “Barack” :op2 “Obama”)
```

- The names themselves are represented as many string constants, linked to a “name” node

Entities

- general concepts are simply stemmed (drop plurality, articles): “the boys” → (b / boy)

“Viacom”

```
(c / company
   :name (n / name :op1 “Viacom”))
```

“Barack Obama”

```
(p / person
   :name (n / name :op1 “Barack” :op2 “Obama”)
```

- This “name” node has a name relation to the concept itself, which is from an ontology or the sentence.

Entities

- general concepts are simply stemmed (drop plurality, articles): “the boys” → (b / boy)

“Viacom”

```
(c / company
   :name (n / name :op1 “Viacom”))
```

“Barack Obama”

```
(p / person
   :name (n / name :op1 “Barack” :op2 “Obama”)
```

- This “name” node has a name relation to the concept itself, which is from an ontology or the sentence.

Entities

- general concepts are simply stemmed (drop plurality, articles): “the boys” → (b / boy)

“Viacom”

```
(c / company
    :name (n / name :op1 “Viacom”))
    :wiki Viacom
```

“Barack Obama”

```
(p / person
    :name (n / name :op1 “Barack” :op2 “Obama”)
    :wiki Barack_Obama
```

- After the main annotation pass, we also add “wikification”, unique IDs for that term.
  (en.wikipedia.org/wiki/ + :wiki label = its page)

Entities

- The ontology is used *only if you do not have a more specific term in the sentence.*
- If a specific descriptor is present, we just use that word instead of finding the closest concept in the ontology.

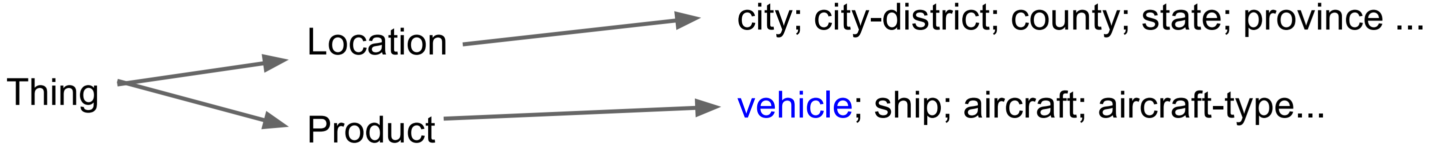
"Ford"

```
(v / vehicle
   :name (n / name :op1 "Ford"))
```

"Ford truck"

```
(t / truck
    :name (n / name :op1 "Ford"))
```

- Ontology is large (100+ types) and hierarchical:


Thing
Location
Product
city; city-district; county; state; province ...
vehicle; ship; aircraft; aircraft-type...

Entities

- There are also special entities that allow us to do very structured annotation of measurable quantities.
- **“Tuesday the 19th”   “five bucks”   “$3 / gallon”**

```
(d/ date-entity
   :weekday Tuesday
   :day 19
```

```
(m /monetary-quantity
   :unit dollar
   :quant 5)
```

```
(r/ rate-entity-91
          :arg1 (m / monetary-quantity
                    :unit dollar
                    :quant 3 )
          :arg2 (v / volume-quantity
                    :unit gallon
                    :quant 1 )
```

Entities

- There are also special entities that allow us to do very structured annotation of measurable quantities.
- **“Tuesday the 19th”   “five bucks”   “$3 / gallon”**

```
(d/ date-entity
   :weekday Tuesday
   :day 19
```

```
(m /monetary-quantity
   :unit dollar
   :quant 5)
```

```
(r/ rate-entity-91
         :arg1 (m / monetary-quantity
                  :unit dollar
                  :quant 3 )
         :arg2 (v / volume-quantity
                  :unit gallon
                  :quant 1 )
```

Designed to be similar to TIMEX normalization

Pronouns

- Pronouns with antecedents in the sentence are just re-entrancies.
- Pronouns without antecedents in the sentence are just the pronoun (made nominative)

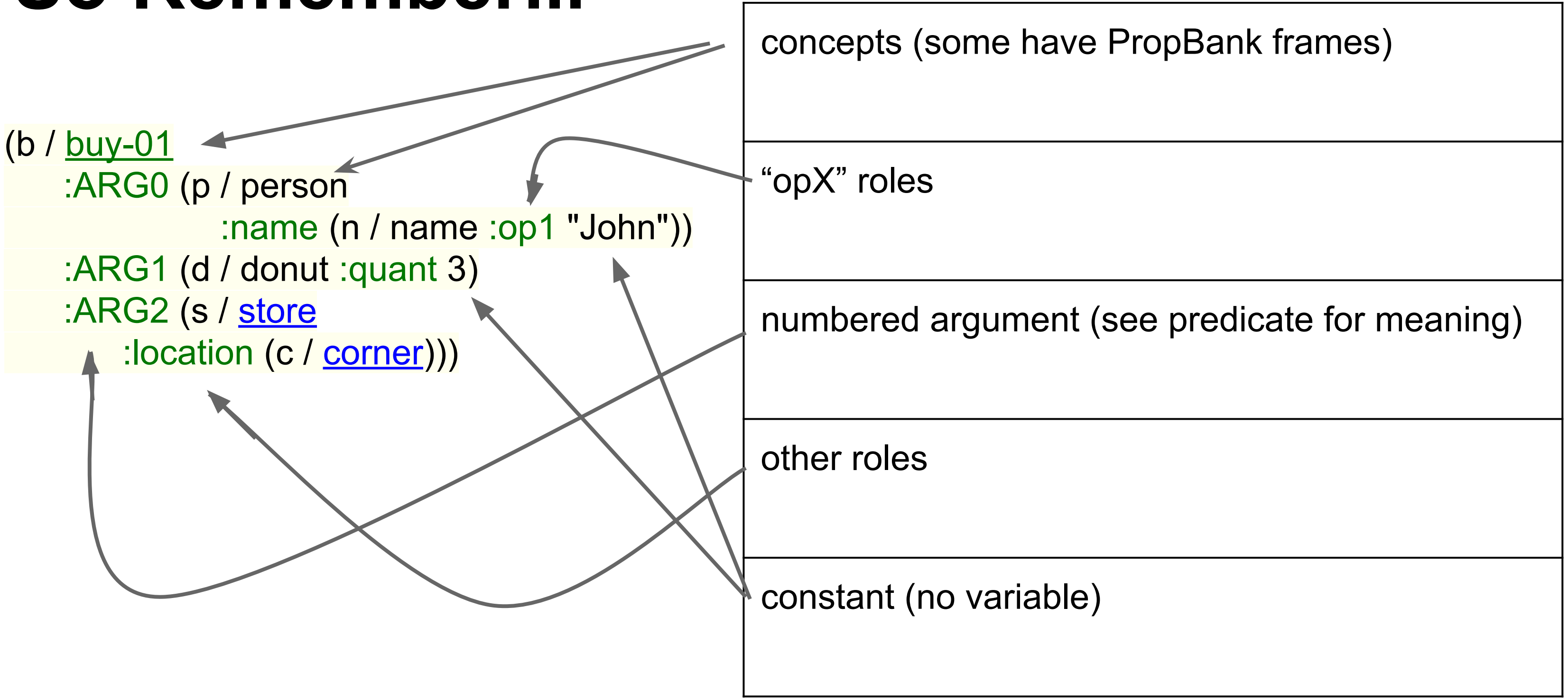
John asked Mary to tutor him

```
(a / ask-02
      :ARG0 (p / person :name (n / name :op1 "John"))
      :ARG1 (t / tutor-01
            :ARG0 p2
            :ARG1 p)
      :ARG2 (p2 / person :name (n2 / name :op1 “Mary”)))
```

Mary was asked to tutor him

```
(a / ask-02
      :ARG1 (t / tutor-01
            :ARG0 p2
            :ARG1 (h / he))
      :ARG2 (p2 / person :name (n2 / name :op1
"Mary")))
```

So Remember...



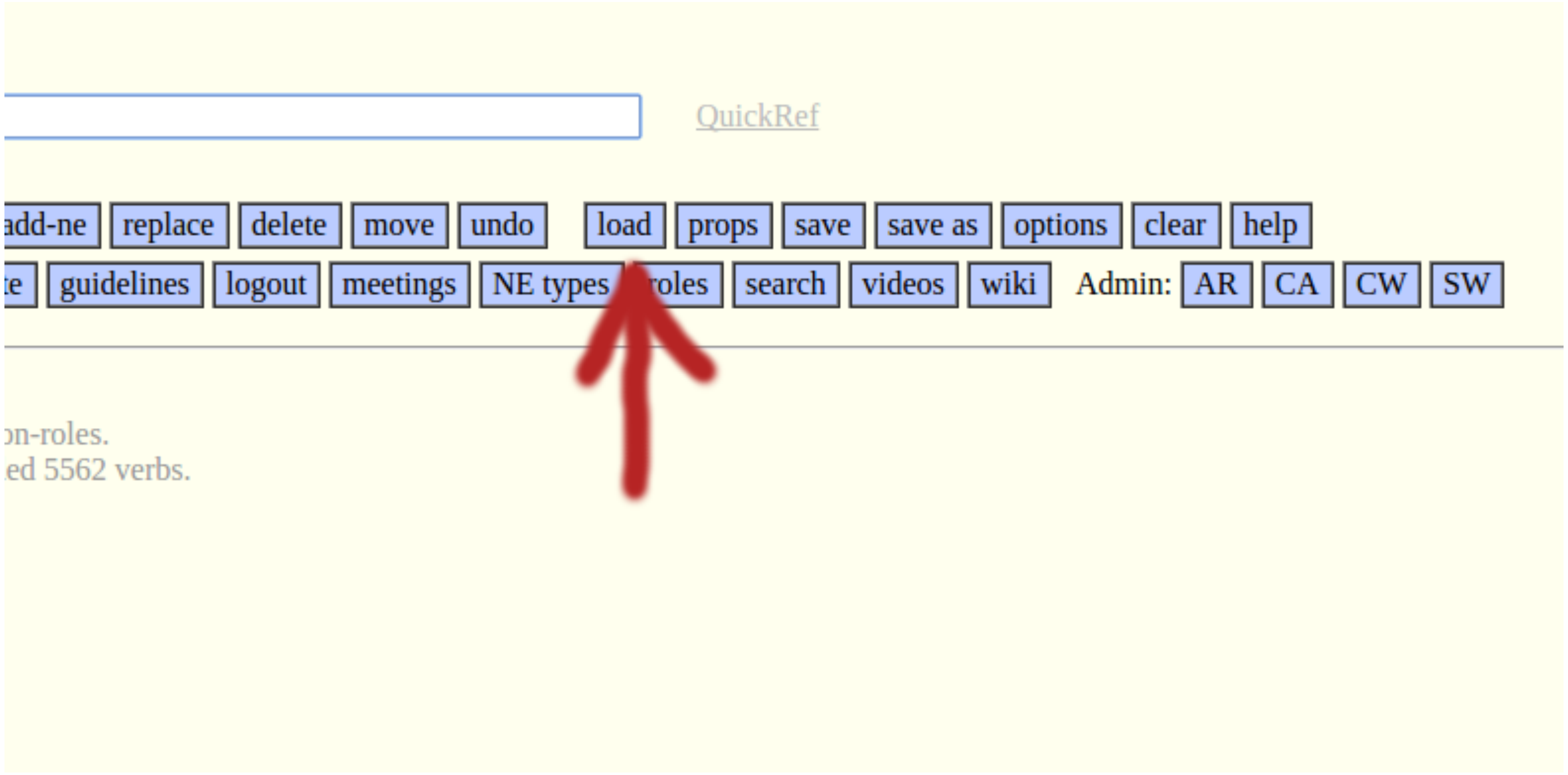
```
(b / buy-01
      :ARG0 (p / person
                 :name (n / name :op1 "John"))
      :ARG1 (d / donut :quant 3)
      :ARG2 (s / store
            :location (c / corner)))
```

concepts (some have PropBank frames)
“opX” roles
numbered argument (see predicate for meaning)
other roles
constant (no variable)

Hands-On Review Examples!

http://tiny.cc/amreditor

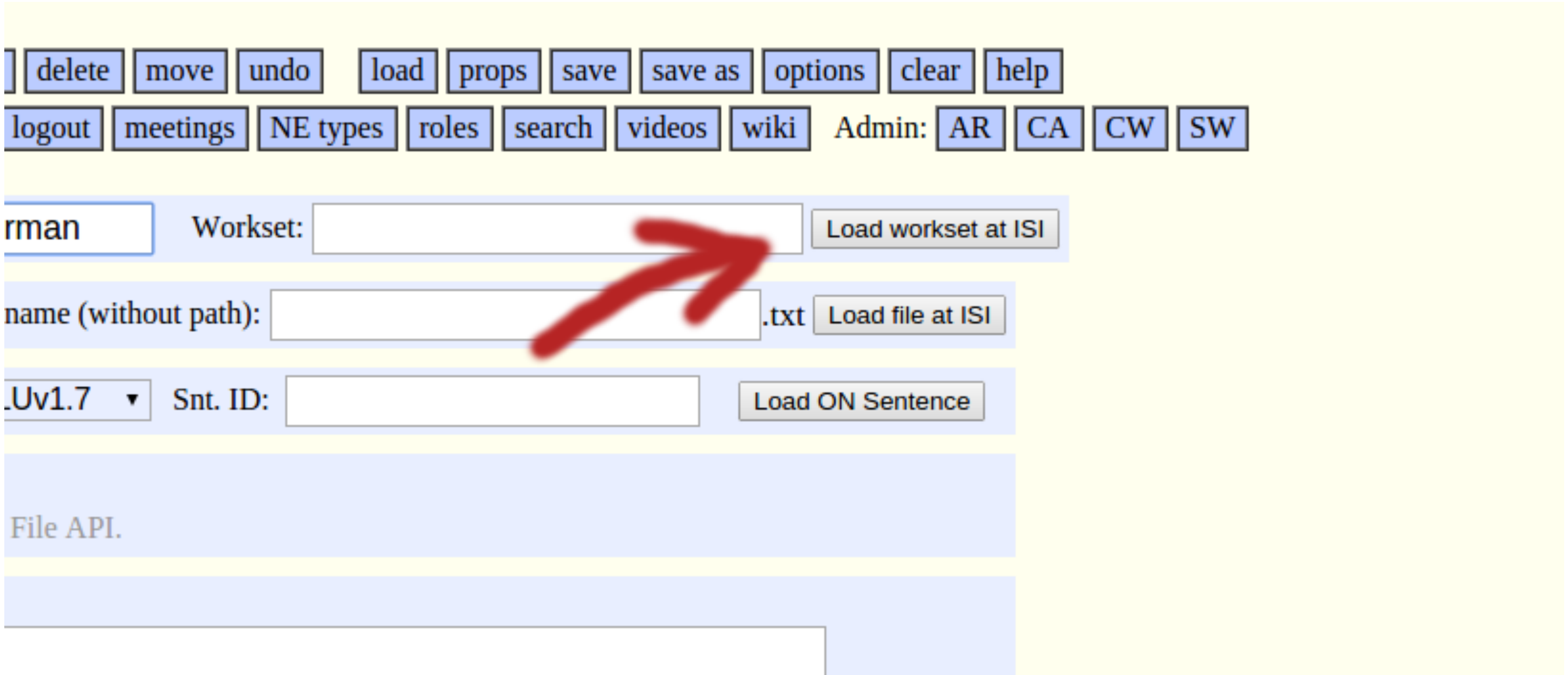
Hands-On Review Examples!



on-roles.
ed 5562 verbs.

Hands-On Review Examples!

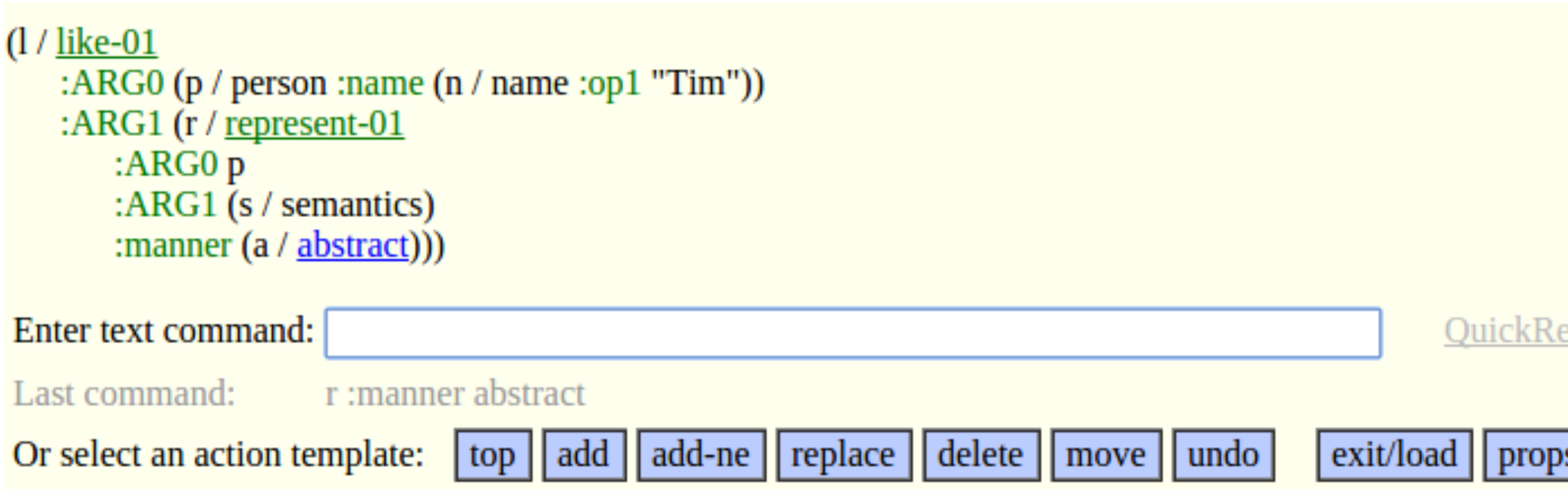
We are loading the “NAACL-tutorial” sentences.



Hands-On Review Examples!

"Tim likes to represent semantics abstractly"

We'll walk slowly through a first sentence.



```
(l / like-01
    :ARG0 (p / person :name (n / name :op1 "Tim"))
    :ARG1 (r / represent-01
        :ARG0 p
        :ARG1 (s / semantics)
        :manner (a / abstract)))
```

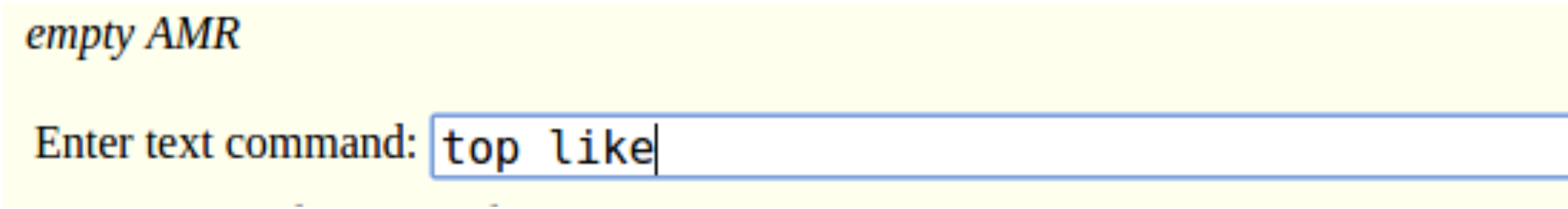
Enter text command: QuickRe

Last command: r :manner abstract

Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

“top” is how to make the root



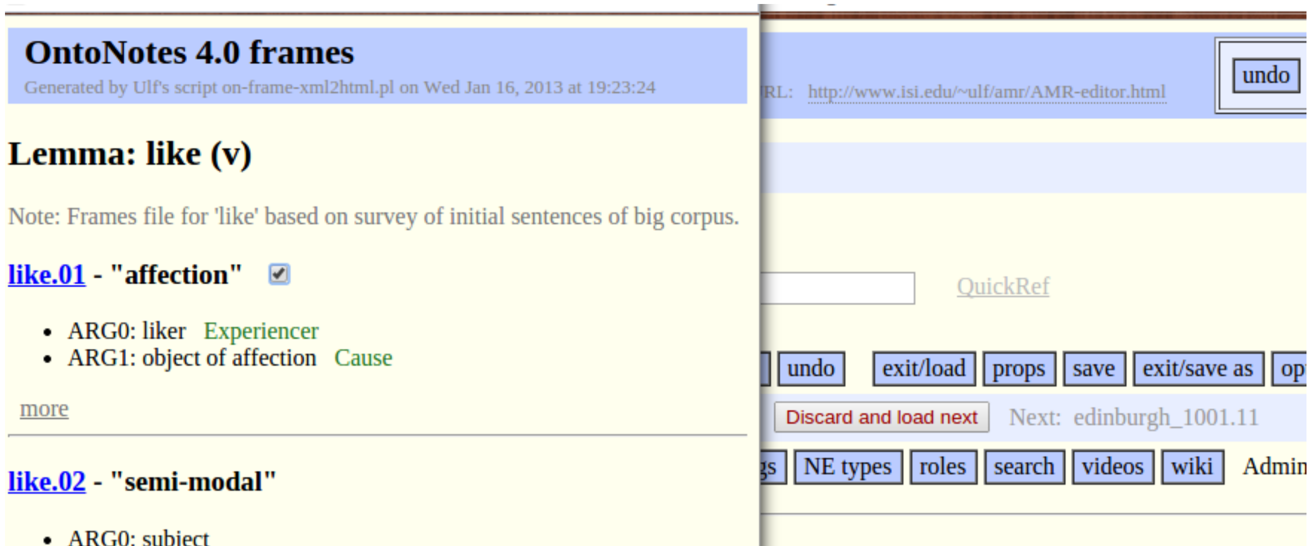
empty AMR

Enter text command: top like

Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

click on “like” to see senses



OntoNotes 4.0 frames

Generated by Ulf's script on-frame-xml2html.pl on Wed Jan 16, 2013 at 19:23:24

Lemma: like (v)

Note: Frames file for 'like' based on survey of initial sentences of big corpus.

like.01 - "affection" ☑

- ARG0: liker Experiencer
- ARG1: object of affection Cause

more

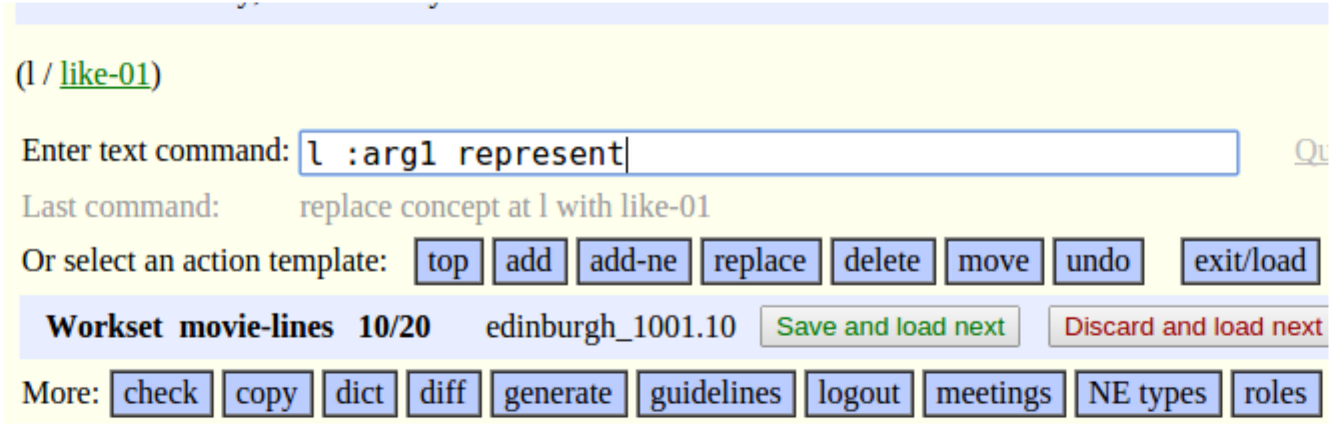
like.02 - "semi-modal"

- ARG0: subject

Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

adding new relations: variable :role concept



(l / like-01)

Enter text command: `l :arg1 represent`

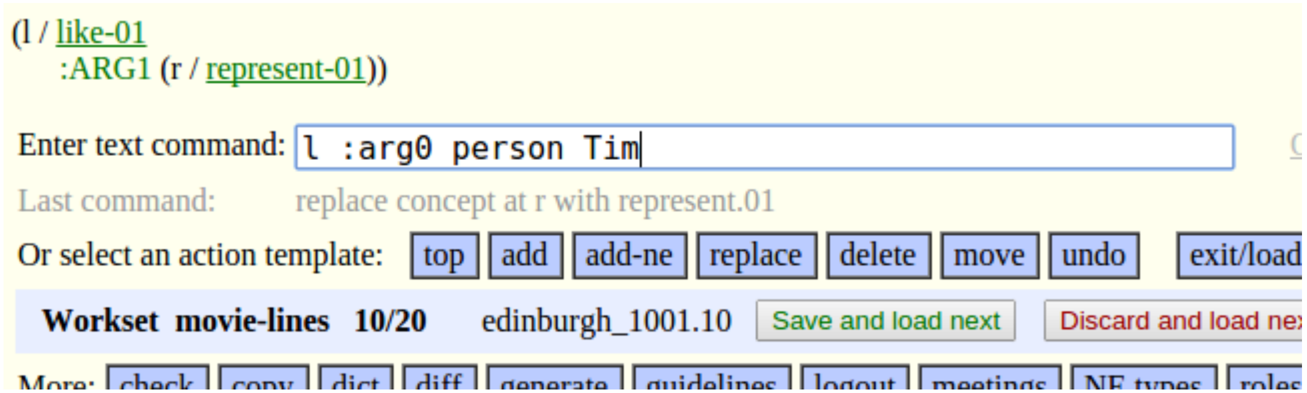
Last command: replace concept at l with like-01

Or select an action template: top add add-ne replace delete move undo exit/load
Workset movie-lines 10/20 edinburgh_1001.10 Save and load next Discard and load next
More: check copy dict diff generate guidelines logout meetings NE types roles

Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

Everything after the third term is automatically added as a name



```
(l / like-01
    :ARG1 (r / represent-01))
```

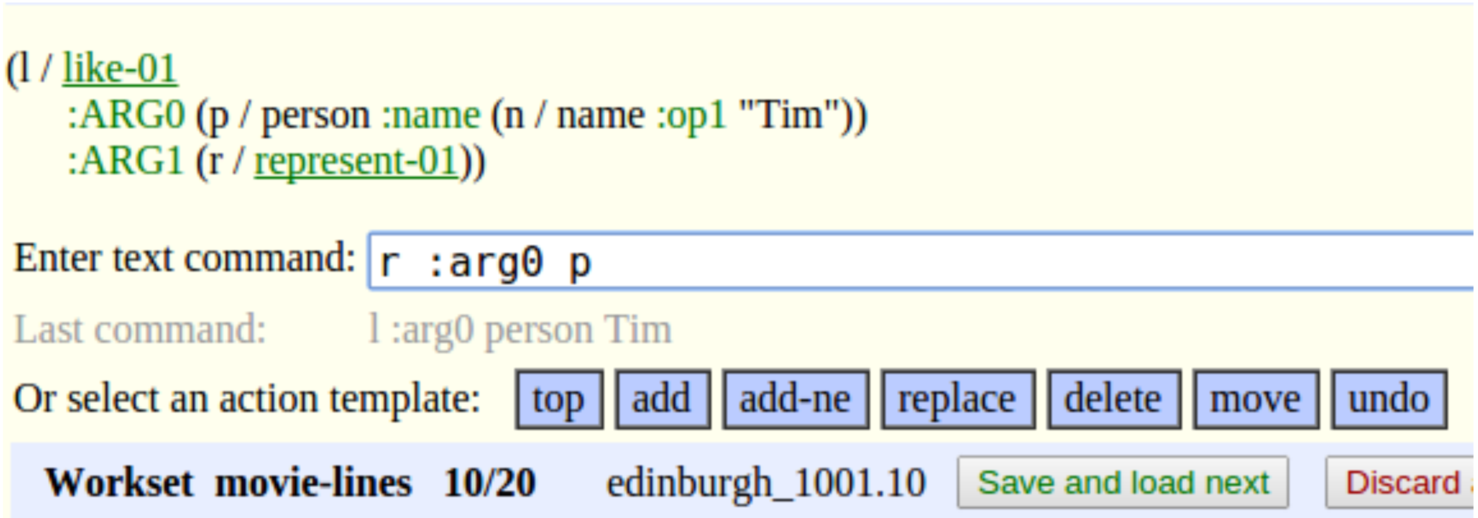
Enter text command: `l :arg0 person Tim`

Last command: replace concept at r with represent.01

Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

Reentrancies are variable :role variable



```
(l / like-01
    :ARG0 (p / person :name (n / name :op1 "Tim"))
    :ARG1 (r / represent-01))
```

Enter text command: `r :arg0 p`

Last command: l :arg0 person Tim

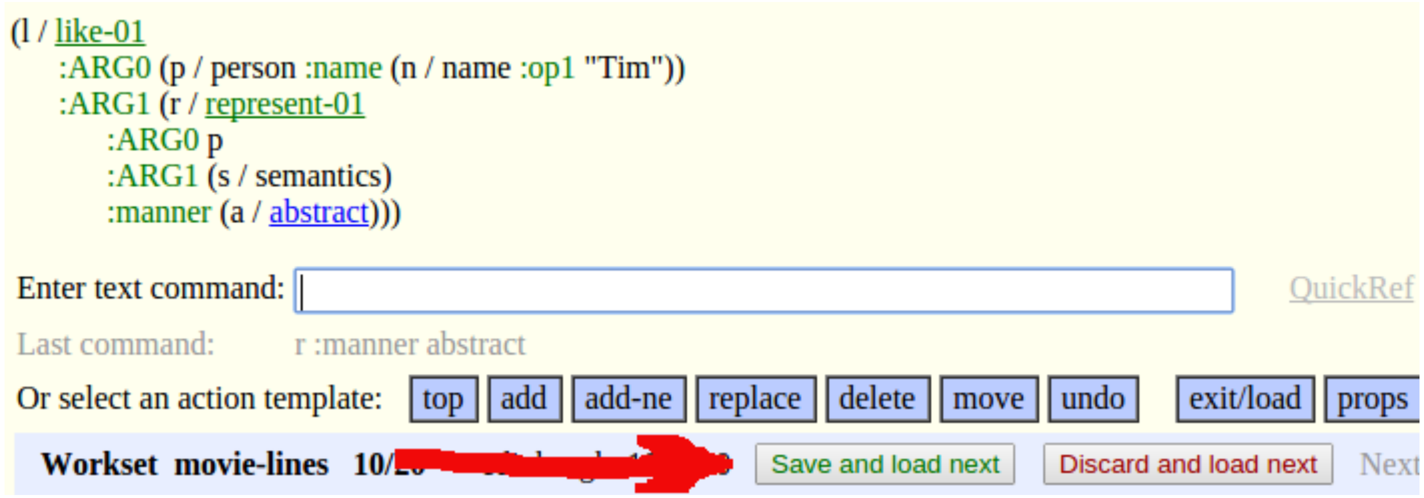
Or select an action template: top add add-ne replace delete move undo

Workset movie-lines 10/20 edinburgh_1001.10 Save and load next Discard

Hands-On Review Examples!

“Tim likes to represent semantics abstractly”

Add the rest and “save and load next”



```
(l / like-01
    :ARG0 (p / person :name (n / name :op1 "Tim"))
    :ARG1 (r / represent-01
        :ARG0 p
        :ARG1 (s / semantics)
        :manner (a / abstract)))
```

Enter text command: QuickRef

Last command: r :manner abstract

Or select an action template: top add add-ne replace delete move undo exit/load props

Workset movie-lines 10/ Save and load next Discard and load next Next

Advanced Topics!

Throwing away light semantics

- For light verbs, copular constructions, and a range of linguistic patterns that are mostly syntactic, we replace them with what they really mean.

"John is nice"

```
(n / nice-41
      :arg1 (p/ person :name (n /name op1 “John”))
```

(We don't allow "to be" at all!)

"John took a bath"

```
(b / bathe-01
      :arg0 (p / person :name (name :op1 “John”))
```

all light verbs convert to
the nearest verbal sense.

Reification replaces “to be”

- Some things involve light semantics, but don’t have their own frame:
  “I think John is at the store.”
- We know what the semantics is doing though! It’s asserting a location that John is at!
- Non-core semantic roles can be converted into predicates using reification. The predicate version of location is “be-located-at-91”
-
```
(t / think-01
    :ARG0 (i / i)
    :ARG1 (b / be-located-at-91
        :ARG1 (p / person :name (n / name :op1 "John"))
        :ARG2 (s / store)))
```

Reification

This allows us to replace “to be” with what is really being claimed:

Joint military exercises are also part of the ICI.

```
(h / have-part-91
    :ARG1 (t / thing
        :name (n / name :op1 "ICI"))
    :ARG2 (e / exercise
        :mod (m / military)
        :mod (j / joint))
    :mod (a / also))
```

It also lets you add modifiers to the semantic roles themselves

The executions are often public and almost always by hanging.

```
(a3 / and
    :op1 (h2 / have-manner-91
        :ARG1 (e / execute-01)
        :ARG2 (p / public)
        :frequency (o / often))
    :op2 (h3 / have-instrument-91
        :ARG1 e
        :ARG2 (h / hang-01)
        :time (a / always
            :mod (a2 / almost))))
```

More Special Predicates

- What about “*John is a pilot for Southwest*”?

We have predicates for organizational and relational predicates!

```
(h / have-org-role-91
      :ARG0 (p / person :name (n / name :op1 "John"))
      :ARG1 (c / company :name (n2 / name :op1 "Southwest"))
      :ARG2 (p2 / pilot))
```

have-org-role.91 - "X is in organization Y as Z"

ARG0: office holder
ARG1: organization
ARG2: office held (title)
ARG3: description of responsibility

Decomposition into concepts

Other contexts for introducing concepts that don’t have words in the data: decomposing complex morphology.

A shoe salesman

```
(p / person
     :ARG0-of (s / sell-01
          :ARG1 (s2 / shoe)))
```

The Indian Government

```
(g / government-organization
     :ARG0-of (g2 / govern-01
          :ARG1 (c / country :name (n / name :op1 "India"))))
```

Decomposition into concepts

Other contexts for introducing concepts that don't have words in the data: decomposing complex morphology.

Notice that we turn pertainyms like "Indian" into the entity they refer to

The Indian Government

```
(g / government-organization
      :ARG0-of (g2 / govern-01
            :ARG1 (c / country :name (n / name :op1 "India"))))
```

Copying

One word can result in multiple predicates!

I ate a sandwich on Thursday and sushi on Friday.

```
(a / and
    :op1 (e / eat-01
        :ARG0 (i / i)
        :ARG1 (s / sandwich)
        :time (d / date-entity :weekday "Thursday"))
    :op2 (e2 / eat-01
        :ARG0 i
        :ARG1 (s2 / sushi)
        :time (d2 / date-entity :weekday "Friday")))
```

Set Operations

We have a predicate “include-91” for sets

I ate five of the 12 donuts” is processed as “I ate five donut out of a set of 12 donuts”

This is useful with our “set” predicate

“I ate 5 of the 12 donuts”

```
(e / eat-01
        :ARG0 (i / i)
        :ARG1 (d / donut :quant 5
                :ARG1-of (i2 / include-91
                        :ARG2 (d2 / donut :quant 12))))
```

include.91 - "subset"

ARG1: subset (or member)

ARG2: superset

ARG3: relative size of subset compared to superset

Set Operations

10% of smokers die of lung cancer.

```
(i / include-91
      :ARG1 (p / person
            :ARG1-of (d / die-01
                  :ARG1-of (c2 / cause-01
                        :ARG0 (c / cancer
                              :mod (l / lung)))))
      :ARG2 (p2 / person
            :ARG0-of (s2 / smoke-02))
      :ARG3 (p3 / percentage-entity :value 10))
```

Of the set of people who smoke….

… 10% of that set …

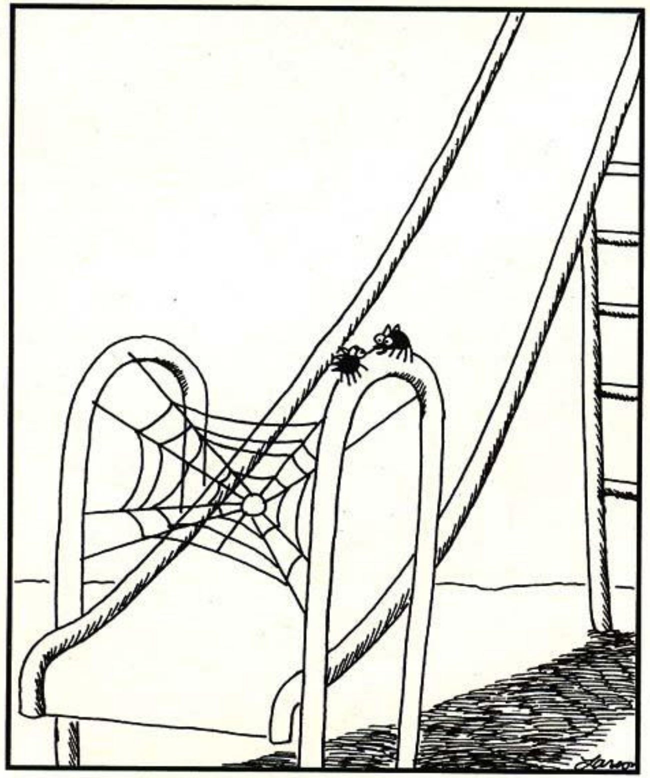
.. are people who die because of lung cancer

Many small additional patterns

- "Hallucinating" a concept usually requires precedent in the AMR dictionary.
- We have patterns for how to handle many specific issues.
- For example, "like" can be "resemble-01":

If we pull this off, we'll eat like kings

```
(e / eat-01
      :ARG0 (w / we)
      :ARG1-of (r / resemble-01
            :ARG2 (e2 / eat-01
                  :ARG0 (k / king)))
      :condition (p / pull-03
            :ARG0 w
            :ARG1 (t / this)))
```



"If we pull this off, we'll eat like kings."

Let’s look at real data

```
(n / need-01
      :ARG0 (w / we)
      :ARG1 (b / borrow-01
            :ARG0 w
            :ARG1 (p / percentage-entity :value 55
                  :ARG1-of (i / include-91
                        :ARG2 (p2 / price
                              :mod (h / hammer))))
            :time (u / until
                  :op1 (p3 / possible
                        :domain (g / get-01
                              :ARG0 w
                              :ARG1 (p4 / permit-01
                                    :ARG1 (p5 / plan-01)
                                    :purpose-of (r / restore-01)
                                    :ARG0-of (a / allow-01
                                          :ARG1 (m / mortgage-01
a.                                                :ARG0 w))))))))
```

We need to borrow 55% of the hammer price until we can get planning permission for restoration which will allow us to get a mortgage

Nearly identical AMRs:
we need a loan for 55% of the hammer price
of the full hammer price, we just need to borrow 55%

Let’s look at real data

```
a.  (n / need-01
        :ARG0 (w / we)
        :ARG1 (b / borrow-01
            :ARG0 w
            :ARG1 (p / percentage-entity :value 55
                :ARG1-of (i / include-91
                    :ARG2 (p2 / price
                        :mod (h / hammer))))
            :time (u / until
                :op1 (p3 / possible
                    :domain (g / get-01
                        :ARG0 w
                        :ARG1 (p4 / permit-01
                            :ARG1 (p5 / plan-01)
                            :purpose-of (r / restore-01)
                            :ARG0-of (a / allow-01
                                :ARG1 (m / mortgage-01
                                    :ARG0 w))))))))
```

We need to borrow 55% of the hammer price until we can get planning permission for restoration which will allow us to get a mortgage

identical AMRs
until such time as we get permission to plan for restoration
up until we get permits for restoration planning

Let’s look at real data

```
(n / need-01
      :ARG0 (w / we)
      :ARG1 (b / borrow-01
            :ARG0 w
            :ARG1 (p / percentage-entity :value 55
                  :ARG1-of (i / include-91
                        :ARG2 (p2 / price
                              :mod (h / hammer))))
            :time (u / until
                  :op1 (p3 / possible
                        :domain (g / get-01
                              :ARG0 w
                              :ARG1 (p4 / permit-01
                                    :ARG1 (p5 / plan-01)
                                    :purpose-of (r / restore-01)
                                    :ARG0-of (a / allow-01
                                          :ARG1 (m / mortgage-01
a.                                              :ARG0 w))))))))
```

We need to borrow 55% of the hammer price until we can get planning permission for restoration which will allow us to get a mortgage

similar AMRs
permits allowing us to get a mortage

Data

- Release 4 has 18,779 AMRs total. (LDC2014E41)
- Release 5 will include :
    - wikification
    - more AMRs
    - more quality control
    - (pretty much) no cycles
- Small (100-AMR) sets of Czech and Chinese AMRs have been annotated (conversion of Prague tectogrammatical annotation is under development)
- PropBank releases will soon all be converted to AMR style (mapping nominalizations to verbs, etc) and re-released.

Comparison - Semantic Roles

AMR: 70+ non-core roles, many verb-sense specific roles (up to 5 args/roleset, more than 10,000 rolesets)

FrameNet: large inventory of frame-specific roles

VerbNet: inventory of thematic roles

Groningen Meaning Bank: VerbNet inventory

Most others: small inventory of roles (agent, theme, etc.)

Comparison - Sense Lexicon

Groningen Meaning Bank: (automatic) WordNet synsets

FrameNet/UCCA: Mark senses by frame/script, not lemma

AMR **/PropBank:** coarse-grained senses (get high ITA)

Prague Dependency TB: valency lexicon rolesets

Most others: undisambiguated concepts as predicates

Comparison - Entities

AMR: Rich named entity ontology (100+ types), wikification

GALE/Ontonotes Annotations: 29 types, 64 subtypes

Groningen Meaning Bank: 7 NE types

Domain-specific (ACE/UMLS/etc.): rich; not all entities

Others: no entity typing

Comparison - Alignment with text

Deepbank; Groningen Meaning Bank: Semantics linked up to a theory of its derivation from syntax (HPSG; CCG)

PropBank, Semantic Treebank: grounded in PTB

Most others: Some link to words in sentence

AMR: No alignment to text (plan to release a few alignments)

Comparison - Logic/Scope/Entailments

Deepbank; Groningen Meaning Bank: Semantics grounds out in logical formalisms (DRT and MRS, respectively)

AMR entailment: linkage between its lexicon and VerbNet may allow rich decomposition

AMR scope: No scope of quantification

Comparison - Size and Quality

AMR: 18,779 sentences, goes beyond newswire, fully manual

Prague Dependency TB: WSJ in Czech and English, manual

Deepbank; Groningen Meaning Bank: Large; automatic parses with human correction/feedback.

UCCA: fully manual, 160k tokens

UNL:

Rich semantic systems with little affiliated data: TMR, LCS, ...

Ancillary Slides

	AMR	*PropBank (PTB/OntoNotes)*	*FrameNet*	*MRS (often =HPSG)*	*GMB & CCGbank*	*Prague CEDT*	*TMR*
Rich Verbal Lexicon	*yes*	*Yes*	*yes*	*no*	*no*	*yes*	*yes*
Semantic Roles	*lexically specified*	*lexically specific*	*frame specific*	*small inventory*	*DRT or FOL*	*small inventory w/ valence lexicon*	*yes*
Entity Ontology	*yes*	*no*	*no*	*no*	*no?*		*yes*
Full-sentence description	*yes*	*no*	*no*	*?*	*no?*		
Align to words	*No*	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*
Project onto syntactic representation?	*no*	*PTBII*	*no*	*HPSG*	*CCG*	*functional dependency syntax*	*HPSG*
Scoped quantifiers	*no*	*no*	*no*	*?*	*?*	*?*	*?*
Lexical coverage on new data	*good coverage*	*strong*	*moderate*	*good unless you need new rules for a new domain...*	*n/a*	*strong (Vallex; now mapped to AMR inventory*	*low (extremely fine-grained lexicon)*
Lexical generalization	stem	stem	network	*?*	*?*	*?*	*?*
Annotated data							